NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

CONTROLLED ACCESS PROTECTION IN THE TELESCRIPT™ PROGRAMMING LANGUAGE

by

Robert Lawrence Marlett

September, 1996

Thesis Advisor:	Cynthia E. Irvine
Second Reader:	Louis D. Stevens

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1996	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE **CONTROLLED ACCESS PROTECTION IN THE TELESCRIPT™ PROGRAMMING LANGUAGE**

5. FUNDING NUMBERS

6. AUTHOR(S) Marlett, Robert Lawrence

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Research on the ability of the Telescript™ language and execution mechanism to enforce controlled access protection on mobile agents moving in and across distributed computer networks has not been published. Nor has General Magic, the creator of the language, conducted security testing on their product.

This thesis investigates whether the mobile agents and execution mechanism proposed by General Magic in its Telescript™ language meet the Class C2 Controlled Access Protection criteria as promulgated in the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC). This was done by conducting an analysis of the documentation provided by General Magic in their Telescript™ Development Kit (TDK) and Active Web Tools™ (AWT).

The results of this thesis show that the mobile agents and execution mechanism of the Telescript™ language do not meet the criteria for TCSEC Class C2 Controlled Access Protection. In particular, the criteria for object reuse, system architecture, system integrity, security testing and security documentation are not met. However, discretionary access control (DAC) can be enforced using a user-defined security policy and the requirements for identification and authentication (I&A) and audit are satisfied.

14. SUBJECT TERMS Remote Programming; Mobile Agents; Discretionary Access Control (DAC).

15. NUMBER OF PAGES 21

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

CONTROLLED ACCESS PROTECTION IN THE TELESCRIPT™ PROGRAMMING LANGUAGE

Robert Lawrence Marlett
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1986

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1996

Author: Robert Lawrence Marlett
____
Robert Lawrence Marlett

Approved by: Cynthia E. Irvine
____
Cynthia E. Irvine, Thesis Advisor

Louis D. Stevens
____
Louis D. Stevens, Second Reader

Ted Lewis
____
Ted Lewis, Chairman
Department of Computer Science

ABSTRACT

Research on the ability of the Telescript™ language and execution mechanism to enforce controlled access protection on mobile agents moving in and across distributed computer networks has not been published. Nor has General Magic, the creator of the language, conducted security testing on their product.

This thesis investigates whether the mobile agents and execution mechanism proposed by General Magic in its Telescript™ language meet the Class C2 Controlled Access Protection criteria as promulgated in the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC). This was done by conducting an analysis of the documentation provided by General Magic in their Telescript™ Development Kit (TDK) and Active Web Tools™ (AWT).

The results of this thesis show that the mobile agents and execution mechanism of the Telescript™ language do not meet the criteria for TCSEC Class C2 Controlled Access Protection. In particular, the criteria for object reuse, system architecture, system integrity, security testing and security documentation are not met. However, discretionary access control (DAC) can be enforced using a user-defined security policy and the requirements for identification and authentication (I&A) and audit are satisfied.

TABLE OF CONTENTS

LIST OF FIGURES

DEDICATIONS

Though it is a small consolation and can never adequately compensate them, I wish to dedicate this thesis to my wife and children for their perseverance and to my parents who taught me the value of education and who never lost their confidence in me. It is my fervent wish that this thesis might, in some small way, reward their confidence and justify their unwavering support.

I. INTRODUCTION

A. PURPOSE

The ability of *mobile agents* to move in and across distributed networks carrying and executing embedded code poses a potential security threat to Department of Defense (DoD) trusted computer systems. As mobile agents act with the same permissions and privileges as their owners, they must be subject to the same methods of identification and authentication that would be imposed upon their owners. Once an *agent* is welcomed on a host computer it must be monitored to ensure that access restrictions are maintained, code execution is non-hostile and system resources are not monopolized. Mobile agents are different from processes that use remote execution in that the agent can perform tasks on its owner's behalf without maintaining continuous communication between itself and the owner. The agent acts independently in effect [Ref. 1].

Before a distributed network is agent-enabled the system administrator and security officer need to understand the operating characteristics and behaviors of mobile agents. Care must be taken to configure the host system, particularly the agent-server, to provide a secure operating environment.

The majority of this thesis is devoted to an examination of General Magic's Telescript™ Development Kit™ (TDK) and Active Web Tools™ (AWT) to determine if they meet the minimum Class C2 requirements for Discretionary Access Control (DAC) as established in the Department of Defense Trusted Computer System Evaluation Criteria. (TCSEC). Whereas both of these applications create agents and run-time environments for them to be interpreted and executed in, the later is being marketed as an agent-enabled web server and is simply an extended version of the former. Both make use of the Telescript™ Programming Language. This is an *object-oriented*, communications-centric language specifically designed for carrying out complex networking tasks: navigation, transportation, authentication, and so on [Ref. 2].

B. BACKGROUND

What is meant by the term Agent? In general the properties of agents can be defined in terms of independence (autonomy), intelligence, communication, learning, mobility and representation of the user [Ref. 3]. In current usage the meaning of agent is fairly broad and in need of refinement. If an agent is simply a program that performs a task for a user, some would argue that a mail *daemon* is an agent while others prefer to think of agents as intelligent actors that traverse networks to negotiate and perform transactions on their behalf. For the purpose of this thesis, an agent will demonstrate following properties:

- autonomous: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
- social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;
- reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of

other agents, the *Internet* , or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;

- pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.[Ref. 4]

The agents created with General Magic's TDK and AWT demonstrate all of these characteristics and are autonomous or mobile in the sense that they can transfer to any Telescript™ enhanced server or client at any time during execution by invoking the *go* command. The agent will then travel, preserving its current state. It may travel to another region on the same machine, to a different machine or to any machine that can be reached by its network and resume execution. This ability is significantly different from using remote procedure calls as the agent does not maintain communications with its source machine and it may carry, within itself, the procedure to be executed remotely. Rather, it is truly independent in its new environment. Telescript™ Agents are significantly more powerful than those found in Sun's Java™ applications. Although Java™ displays a level of portability in that it can be moved from one environment to another for execution, Java™ programs or applets do not have the ability to move themselves. A Java™ program is moved from machine to machine by cooperating higher-level programs (servers and *browsers*); Java™ programs operate as tightly controlled subsystems of those higher-level programs [Ref. 3].

C. SCOPE, LIMITATIONS AND ASSUMPTIONS

This thesis will focus on the Telescipt™ and the General Magic paradigm of mobile agents. Their agents represent one of the first commercially available agents of this sort and appear to be the most capable on the market.

D. RESEARCH OVERVIEW

This section provides an overview of each chapter's contents.

1. Introduction

2. The Telescript ™ Mobile Agent Paradigm

Chapter II describes the mobile agent paradigm as advanced by General Magic in their products. The creation of these agents, their actions, roles, and operating environment will be described in detail.

3. Controlled Access Protection

Chapter III discusses the Discretionary Access Requirements (DAC) as promulgated by the National Computer Security Center to meet the Department of Defense Trusted Computer System Evaluation Criteria.

4. Controlled Access Protection in Telescript™

Chapter IV evaluates whether the agents and operating environment as advanced by General Magic meet the discretionary access goals for Criteria Class C2.

5. Conclusion

Chapter V provides a summary of the initial research findings and offers recommendations for future research.

II. THE TELESCRIPT™ PARADIGM ON MOBILE AGENTS

A. BASICS

The Telescript™ paradigm emerged from a conscious effort by General Magic to improve the efficiency of computer communications networks. Currently the central organizational principle of computer communications networks is remote procedure calling (RPC) [Ref. 5, pp. 561-570]. The RPC paradigm uses computer-to-computer communications to enable one computer to call procedures in another. Each message that the network transports either requests or acknowledges a procedure's performance. A request includes data that are the procedure's arguments. The response includes data that are its results. The procedure itself is internal to the computer that performs it. Two computers whose communication follow the RPC paradigm agree in advance upon the effects of each remotely accessible procedure and the types of its arguments and results. This agreement constitutes a *protocol* [Ref. 1, p. 3].

A user or client computer with work to perform on a server completes the task through a series of remote procedure calls. Each call involves a request sent from user to server and a response sent from server to user. The salient characteristic of remote procedure calling is that interaction between the user computer and the server entails two acts of communication, one to ask the server to perform a procedure, and another to acknowledge that the server did so or to return the results of the computation. Thus completion of the task requires ongoing communication. When the task is synchronous at the client, time is wasted in the wait state.[Ref. 1, p. 4]

> An alternative to remote procedure calling is *remote programming* (RP). The RP paradigm views computer-to-computer communication as enabling one computer to not only to call procedures in another, but also to provide the procedures to be performed. Each message that the network transports comprises a procedure that the receiving computer is to perform and data that are its arguments. In an important refinement, the procedure is one whose performance the sending computer began or continued, but that the receiving computer is to continue; the data are the procedure's current state.[Ref. 1, p. 4]

> Two computers whose communication follows the RP paradigm agree in advance upon instructions that are allowed in a procedure to be sent by the user and the types of data that are allowed in its state. Their agreements constitute a *language*. The language includes instructions that let the procedure make decisions, examine and modify its state, and, importantly, call procedures provided by the receiving computer. Such procedure calls are local rather than remote. The procedure and its state are termed a *mobile agent* to emphasize that they represent the sending computer even while they are in the receiving computer.[Ref. 1, p. 4]

The striking characteristic of remote programming is that a user computer can perform a task on a server without continuous interaction once the network has transported an agent between them [Ref. 1, p. 4].

B. MOBILE AGENT CONCEPTS

One of the first commercial implementations of the mobile agent concept is General Magic's Telescript *TM* Technology which allows automated as well as interactive access to a network of computers. The focus of this technology is the Internet-based electronic marketplace.

Telescript™ technology is implemented with the following principle concepts: *places*, *agents*, *travel*, *meetings*, *connections*, *authorities* and *permits*.

1. Places

Any network of computers, regardless of size, is a collection of places. The sum of all places under the same authority (the individual or organization that a place or agent represents) is termed a *region*. Places offer services to mobile agents that enter them. Servers and clients can both contain one or more places. On each individual machine in the network an *engine-place* contains all other places which may contain sub-places. A collection of different machines, each with its own engine-place and associated sub-places, under the same authority is referred to as a *cloud*. See Figure 1 on the next page for a pictorial representation of an engine-place, region and cloud.

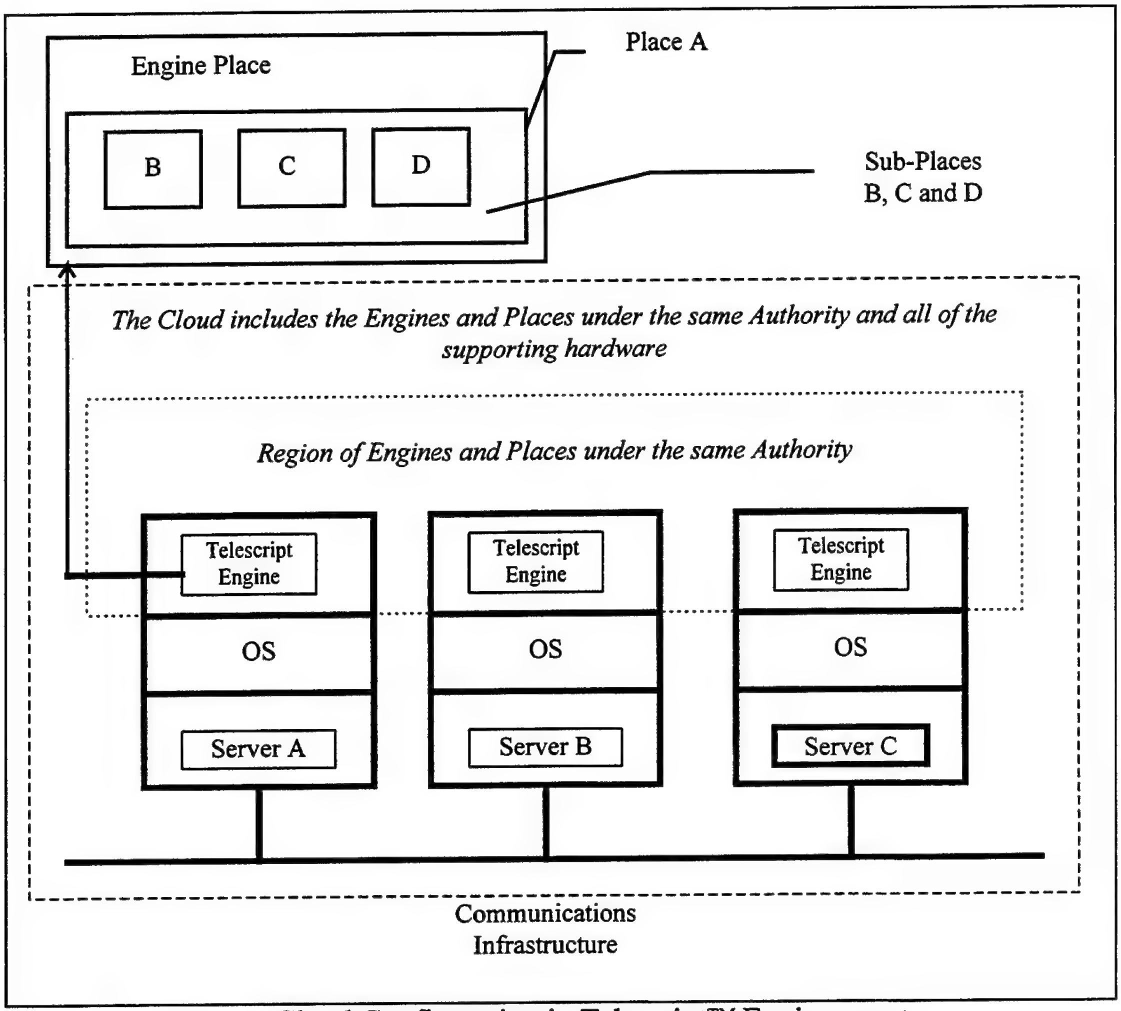
Place A
Engine Place
B
C
D
Sub-Places
B, C and D
The Cloud includes the Engines and Places under the same Authority and all of the supporting hardware
Region of Engines and Places under the same Authority
Telescript Engine
OS
Server A
Telescript Engine
OS
Server B
Telescript Engine
OS
Server C
Communications
Infrastructure

Figure 1. Cloud Configuration in Telescript™ Environment

2. Engine

The engine is a software program or process that implements the Telescript™ Language by maintaining and executing places within its control, as well as the agents that occupy or enter those places. Normally, there will be only one engine per server.

Conceptually, the engine is a virtual-machine that runs on top of its host computer's operating system. The engine resembles an operating system in that it performs process and memory-management, coordinates resources and mediates system-calls. These similarities will be described in detail later.

> At least conceptually the engine draws upon the resources of its host computer through three *application program interfaces (APIs).* A storage API allows the engine to access the nonvolatile memory it requires to preserve place and agents in case of computer failure. A transport API allows the engine to access the communication media that it requires to transport agents to and from other engines. An external applications API lets the parts of an application written in the Telescript™ language to interact with those written in C.[Ref. 1, p. 14]

3. Agents

An agent is a process that has the ability to travel between places. Agents can perform data collection, conduct transactions or interact with other agents. Each agent can inhabit only one place at a time but can move from one place to another, thus occupying different places at different times. Agents are independent in that their procedures are performed concurrently [Ref. 1, p. 7]. A client may launch multiple agents and conversely multiple agents may be active in any place.

4. Travel

The ability of an agent to move from one place to another is termed 'travel' and is the resonant feature of a remote programming system. The Telescript™ Language allows a computer to package an agent, i.e. its *procedures* and *state*, so that it can be transported to another computer. The agent itself decides when to travel using the *go* instruction. This instruction requires a *ticket*, data that specify the agent's destination and other terms of the trip. If the trip is successful the agent is unpacked and its next instruction is executed at its new destination. In effect, networking is reduced to a single instruction. It is important to note however, that it is not necessary or desired for all agents to travel. In particular, non-mobile server agents are assigned to places to coordinate the entry of user agents and facilitate the use of server services or data.

5. Meetings

Any two agents that *occupy* the same place can meet. A meeting allows the agents to call each other's procedures. All meetings are mediated by the engine-process. Through meetings, transactions are conducted. Using the *meet* instruction one agent can query another. This meeting requires a *petition* that states the terms under which a meeting can be conducted. These meetings form the core of the electronic marketplace envisioned by General Magic. In this marketplace, user agents roam the networks in search of either service or data and accomplish this through meeting with server agents programmed to facilitate these services.

6. Connections

A connection allows two agents in two different places to make a connection between themselves. Use of connections can enable two agents to communicate in a manner similar to RPC but this would nullify the principle benefit of agents: reduced communications traffic. This feature is currently not available in the current version of Telescript™.

7. Authorities

The authority of an agent or a place is an attribute that represents the individual or organization of the agent or place. Agents and places can discern but can neither withhold nor falsify their authorities. The authority of an agent is verified whenever an agent travels from one region to another. A region is a collection of places provided by computers that are all under the operation of the same authority. If an agent's authority cannot be validated at the source destination the agent is denied entry.

To determine an agent's or place's authority, an agent or place executes the *name* instruction. This instruction returns a *telename* which denotes the entity's identity as well as its authority. Identities distinguish agents or places of the same authority.

8. Permits

Authorities can limit what agents and places can do by assigning permits to them. The permit is data that grants capabilities and is an attribute of both places and agents. An agent or place can discern its own capabilities and that of others but cannot increase them. In addition an agent that can create other agents can't grant those agents capabilities that it does not possess itself.

Permits are used to grant two types of capabilities. One type of permit is used to grant the right to execute certain types of instructions. An example of this is the ability to create other agents. A permit can also be used to limit the amount of system resources an agent or place can consume.

An agent's effective permit is re-negotiated whenever it travels to another region or place within a region.

In most cases the default permit does not explicitly deny the use of most functions of the language and restrictions must be defined explicitly. The primary stated purpose of the permits is not for security per se but to prevent processes from monopolizing server and communications system resources [Ref. 9, p.78].

C. TELESCRIPT™ MOBILE AGENT TECHNOLOGY

1. Language

The Telescript™ programming language permits the developers of communications applications to write the algorithms that the agent will follow and what information it will carry as it traverses a network. Entire applications can be written in the language but typically applications are partially written in C/C++. The portions written in C/C++ are normally used for the agent-to-user interface and software in servers that allow places to interact with databases. The Telescript™ language has the following qualities:

- *Complete.* The language can be used to express any algorithm. Agents can be programmed to make decisions, handle exceptions, and to gather, organize, create and modify information.
- *Object-oriented.* The programmer defines classes of information, one class inheriting the features of others. Classes of a general nature, such as Agent, are predefined by the language. Classes of a specialized nature, such as Shopping Agent, are defined by communicating application developers.
- *Dynamic.* An agent can carry an information object from a place in one computer to a place in another. Even if the object's class is unknown at the destination the object continues to function: its class goes with it.
- *Persistent.* Wherever it goes, an agent and the information it carries, even the program counter marking its next instruction, are safely stored in nonvolatile memory. Thus, the agent persists despite computer failures.
- *Portable and Safe.* A computer executes [interprets] an agent's instructions through a Telescript™ engine, not directly. An agent can execute in any computer in which an engine is installed, yet it cannot

> access directly its processor, memory, file system, or peripheral devices. This helps prevent viruses.
>
> - *Communications-centric*. Certain instructions in the language, several of which have been discussed, let an agent carry out complex networking tasks, such as transportation, navigation, authentication, access control, and so on.[Ref. 1, pp. 13-14]

The language demonstrates a high level of transparency for the application developer. The agents do not have to know where resources are located and can operate on any system without direct knowledge of the system (portability) provided a running Telescript™ engine process is in place.

2. Protocols

> The Telescript™ protocol suite enables two engines to communicate. Engines communicate in order to transport agents between them in response to the 'go' instruction. The protocol suite can operate over a wide variety of transport networks, including those based on the TCP/IP protocols of the Internet, the X.25 interface of the telephone companies, or even electronic mail. The Telescript™ protocols operate at two levels. The lower level governs the transport of agents, the higher level their encoding and decoding. Loosely speaking, the higher-level protocol occupies the presentation and application layers of the seven-layer Open Systems Interconnection (OSI) model.[Ref. 1, pp. 13-14]

The Telescript™ *encoding-rules* specify how an engine encodes an agent, its procedure and state, as binary data and will sometimes omit portions of it to optimize performance. These omitted portions are the lessor objects of the *Agent* class which the individual agent does not possess. Although engines are free to maintain agents in different formats for execution, they must employ a standard format for transport. Agents are encoded and decoded by all engines using the same rules.

> The Telescript™ platform interconnect protocol specifies how two engines first authenticate one another (for example, using public key cryptology) and then transfer an agent's encoding from one to the other. The protocol is a thin veneer of functionality over that of the underlying transport network.[Ref. 1., pp. 15-16]

D. THE TELESCRIPT™ ENGINE AS A VIRTUAL MACHINE

1. Telescript™ as an Extended (Virtual Machine) and Resource Manager

As the Telescipt™ engine and engine-place represent a virtual-machine running atop the native operating system of the host computer operating system terminology can be used to describe its operating characteristics.

To effect the virtual-machine a script is used to *bootstrap* an engine-place object to represent the running Telescript™ engine. The engine-place is used to set up runtime policies, to include security, and provides the environment to instantiate Telescript™ processes which correspond to *threads* in common operating system vernacular.

Each engine maintains an engine-place at a minimum which represents the engine itself and one or more virtual-places that will occupy and execute within the engine place.

In particular,

The engine parses (reads) tokens to create objects, then executes the objects to build complex objects and perform actions. By saying Telescript™ is an interpreted, object-orientated language you are saying the language's engine parses its tokens and executes its objects.[Ref. 7, p. 123]

2. Processes

a. *The Process Model*

In the Telescript™ process-model a process is a named object capable of performing instructions as a self-contained unit. There are two kinds of processes, agent and place, which interact by means of *features*. Features refer to object attributes and operations. These features enable one process to convey to another a reference to, a copy of, or ownership of an object.

Upon the direction of the engine a process (thread) moves through three distinct phases. In the *initialization* phase, the engine requests a potential process perform an *initialize* operation. If successful (no exceptions thrown) the process enters a *live* phase instigated by the engine. The *live* method of a process is simply the executable code embedded in it. To begin the live phase the engine requests the object's *live* operation. Completion of a process' live phase, or an uncaught exception causes a process to enter the *termination* phase which marks the end of that process' execution.

A process can have any number of concurrent activation's and thus any number of execution threads. The engine activates a process to perform any sponsored operation that the engine requests of a process (for example, the 'entering' or 'meeting', as well as the 'live' operation). The

> activation ceases when the performance ends, whether successfully or unsuccessfully.[Ref. 8, p. 10]

A note of caution; realistically, the number of threads is limited by system resources. The maximum number of threads that can be executed before system performance becomes an issue is unknown.

A process (thread) occupies a place within the engine place by *entering* it. The process can enter a place by being constructed by the place itself or by a non-mobile agent that resides within the place. Additionally, upon successful completion of the *go* operation , an agent occupies a place by taking a trip there.

In all cases the engine mediates the entry of a process (thread) to a place by requesting the entering operation of the place. If the operation is successful entry occurs. Otherwise entry fails and the place's existence can be hidden from or revealed to the requesting process at the discretion of the place whose entering operation was invoked.

The engine does not serialize the entry of processes to a place. However, places can provide their own serialization and limit the number of processes that are active within them as well.

The Telescript™ engine can maintain multiple processes called places and agents. Each process has its own stack and thread of execution. The engine will switch among the threads continuously with the effect that the processes appear to execute concurrently.

The Telescipt™ engine uses a queue to accumulate event signals between processes and uses various operations to manipulate them. A priority event can be placed at the head of the queue.

b. Implementation of Processes

The engine must request an entering operation as defined by Telescript™ *Class* place before an agent or place can occupy a place. Access to a place's features do not require the meeting operation whereas any interaction between agents requires use of the meeting operation. In all cases, entering or meeting is mediated by the engine.

c. Interprocess Communication

An agent cannot interact with another agent remotely. They must occupy the same place to meet and *part.* Two meeting agents occupying the same place can conduct a meeting by using the meet operation to gain a reference to the other. Either agent can call the part operation which causes the engine to void the references between them thus ending the meeting.

> Processes can also interact with one another by using event signals. For each process the engine maintains a queue that holds the assigned event definitions of all events the process is enabled to receive.[Ref. 7, p. 211]

d. Process Scheduling

The Telescipt™ engine provides multi-tasking by executing the live method of each agent or place independently. The engine schedules the independent threads and switches preemptively among them favoring no one process unless that process has been granted priority. Threads that are blocked are not scheduled.

Objects can be dedicated or shared resources to enable threads to perform procedures without interruption from other threads. The engine will delay or reschedule threads that are competing for unshared resources.

3. Memory Management

a. Creation of Objects

Every object is owned by a process and a process owns itself. Ownership of an object grants capabilities to the process that owns it. Owned objects can be copied, modified and deleted. Ownership of objects can also be transferred between processes.

> Some operations, including the 'initialize' operation of all objects, and the 'live' methods for agents and places don't have to be explicitly requested. They are requested by the engine automatically...initially when object is instantiated, and 'live', as soon as an agent or place is successfully initialized.[Ref. 7, p. 128]

b. Destruction of Objects

A process can destroy an object that it owns. If a process wishes to destroy an object the engine voids all references to it and its properties. When a process is destroyed or ends its live method (finishes execution) all objects it owns are destroyed and all references to its objects are voided.

Memory space is reclaimed when released. The engine performs garbage collection on the Telescript™ Environment Data Store (secondary media) during periods of low system inactivity. This data store is described greater detail in the next section.

4. Files

This section specifically refers to the files maintained by the Telescript™ engine to perform system initialization and configuration, and system restoration or backup.

a. Object Persistence

The Telescript™ engine uses a *file-store* located on the host's secondary media to enable object persistence by storing them in an *object-oriented database*. Objects will survive as long as there are references to them. In the case of system shutdown or failure the environment is preserved as the engine takes a *snapshot* of the state of all objects and processes at intervals determined by the region authority for the engine. This persistence is not absolute. Although the objects and processes and any communication traffic to and from the engine are committed to the database for restoration and playback, scripts (processes) that are using external methods (interaction with processes outside of the Telescript™ environment) or the host file system are subject to failure due to the non-continuous commit strategy.

b. Object Restoration

Objects and processes can be restored with the object-oriented file store committed to secondary storage subject to the limitations discussed above. System backup and restoration is subject to the policy of the region authority.

5. System Calls/API's/External Applications

a. API's

The Telescript™ engine draws upon the resources of its host computer through three application programming interfaces (APIs). The *storage* API provides access to the computer's non-volatile storage, which the engine uses to preserve places and agents in the event of system failure. The *transport* API provides access to the host computer's communication media by which the engine sends agents to and receive agents from other engines. The *external-applications* API lets the processes of the Telescript™ Environment interact with non-Telescript™ applications or processes of the host computer.[Ref. 8, p. 3]

As the engine is designed to support places and agents of different authorities it has privileged escapes from the language (available only to the authority of the engine) to construct operational, administrative, and managerial (OA&M) tools that are external to the engine.

b. Using system resources and peripheral devices

Use of system resources and peripheral devices is mediated by the engine and access is given only to processes of known authority possessing the proper permits. All interaction with the host system is through the external-applications API which is controlled by the engine itself.

c. External Applications

One of the primary functions of external methods is to transfer and translate between data within the Telescript™ environment and data outside of that environment. Interaction with a host database is one example.

6. The Shell - Thumper™ (Engine Control Panel)

The *Thumper™* acts as the Telescipt™ cloud's command console. It can be used to monitor the activity of all engines within a cloud, restrict or add subscriber privileges, modify the behavior of special facilities and services with a cloud and submit custom Telescript™ scripts to an engine [Ref. 9, p. 5].

a. Starting and Controlling the Engine

The engine is started as a process on the host system through an application interface and is controlled by the Thumper™ as outlined above.

b. Engine Operations

Individual engines can be started and stopped using the Thumper™. In addition, engines and engine-spaces can be reconfigured from the Thumper™ with some limitations. Some changes require the re-initialization of the engine from a *cold-start*.

III. CONTROLLED ACCESS PROTECTION

A. INTRODUCTION

This chapter provides the minimal requirements for a system to be assigned a Class C2 (Controlled Access Protection) rating for enforcing discretionary access control. Most of the quoted material in chapter is taken from the Department of Defense Trusted Computer System Evaluation Criteria, hereafter referred to as the "Criteria" [Ref. 10]. In each section the requirements may be followed by amplifying remarks and examples intended to clarify the requirements.

B. SECURITY POLICY

1. Discretionary Access Control

> The TCB shall define and control access between named users and named objects (e.g., files and programs) in the ADP system. The enforcement mechanism (e.g., self/group/public controls, access control lists) shall allow users to specify and control sharing of those objects by named individuals, or defined groups of individuals, or by both, and shall provide controls to limit propagation of access rights. The discretionary access control mechanism shall, either by explicit user action or by default, provide that objects are protected from unauthorized access. These access controls shall be capable of including or excluding access to the granularity of a single user. Access permission to an object by users not already possessing access permission shall only be assigned by authorized users. [Ref. 10, p. 15]

A TCB or Trusted Computer Base is the sum of all protection mechanisms in a computer system to include the hardware, firmware and software and is responsible for enforcing a security policy.

Discretionary access control is a method of restricting access to *objects*, typically files and directories, based on the identity of *subjects*. Subjects are the active entities in the computer system normally consisting of programs acting on behalf of the user and representing the user's themselves. Examples of subjects include a user entering commands at the command-prompt or a word processor application that is attempting to open a file. DAC is discretionary in that it permits the subjects to make their objects accessible to others leaving the implementation of access controls to the TCB. Users can change access control lists for their objects or pass capabilities to other users for access to their objects. Users may also make new objects by copying them and can grant access to the new objects to a new set of subjects.

At the C2 Class level of access protection, control of newly created objects must not automatically default to public access. Rather, public access to newly created objects could be granted using a mask that had been set up in advance by the user. The system must have specific rules that limit access to an object to its creator or to a previously specified set or subset of users or the system must require the creator of an object to specify who will be allowed to access it before he is allowed to create the object. Limiting access to the creator of the object is the preferred method [Ref. 11, p. 21].

> The mechanism [DAC] must have the ability to include or exclude access on a per user basis. If group access controls are provided, groups must be precisely defined by listing unique names of users in the groups. Hence, groups are made of named users. [Ref. 11, p. 21]

Named users and objects are uniquely identified to the TCB, i.e. user name, or file name, and are used by the DAC mechanism to perform access control decisions.

2. Object Reuse

> All authorizations to the information contained within a storage object shall be revoked prior to initial assignment, allocation or reallocation to a subject from the TCB's pool of unused storage objects. No information, including encrypted representations of information, produced by a prior subject's actions is to be available to any subject that obtains access to an object that has been released back to the system. [Ref. 10, p. 15]

This requirement means that objects created by a subject in any storage medium (main memory, secondary storage, etc.) should not be recoverable when destroyed by the user. For example, if an object in memory is de-referenced it is not completely destroyed by the elimination of all references to it. The actual media must be overwritten so that the original object cannot be reconstructed. The deletion of a file in a DOS based file system is a familiar example where the object is not completely destroyed. Deleting a file simply removes its reference in the file access table. To be completely destroyed and eliminate the possibility of reconstruction the physical space occupied by the file must be overwritten.

C. ACCOUNTABILITY

1. Identification and Authentication (I&A)

> The TCB shall require users to identify themselves to it before beginning to perform any other actions that the TCB is expected to mediate. Furthermore, the TCB shall use a protected mechanism (e.g., passwords) to authenticate the user's identity. The TCB shall protect authentication data so that it cannot be accessed by any unauthorized user. The TCB shall be able to enforce individual accountability by providing the capability to uniquely identify each individual ADP system user. The TCB shall also provide the capability of associating this identity with all auditable actions taken by that individual. [Ref. 10, p. 16]

If the basis of access to objects in a system is based on the identity of a subject be it a user or a group of identified users it is reasonable that a user should be known to the system through some means of identification, normally a user name or group ID. Additionally the system should be able to authenticate or verify that the users are who they claim to be. This is normally done with a passwords, personal information numbers (PINs), or like information that can be provided only by the user.

> I&A must distinguish operators, system administrators, and system security officers from ordinary users in order to record security related events as actions initiated by the individuals performing those roles. Since individuals performing those roles may also be ordinary users of the system, it's necessary to distinguish the people when acting as ordinary users.[Ref. 12, p. 13]

Ordinary users should not be able to perform security actions nor access the database that the DAC mechanism uses to authenticate users. Users who perform security functions should be able to log-in with an ID and password that is distinct from those used when they are acting as ordinary users.

"For identification/authentication events audits the origin of the request (e.g., terminal ID) shall be included in the audit" [Ref. 12, p12]. However, if the identification/authentication event is across networks, the terminal ID will not be available and it must be possible to trace backwards through the network components to determine the origin of the *login*.

All actions taken by a user must be subject to an audit which matches the process action with the ID of the user. All parent and child processes of a user must be attributable to that user for audit purposes. That is to say, all process IDs must be uniquely related to a user ID.

2. Audit

> The TCB shall be able to create, maintain, and protect from modification or unauthorized access or destruction an audit trail of accesses to the objects it protects. The audit data shall be protected by the TCB so that read access to it is limited to those who are authorized for audit data. The TCB shall be able to record the following types of events: use of identification and authentication mechanisms, introduction of objects into a user's address space (e.g., file open, program initiation), deletion of objects, and actions taken by computer operators and system administrators and/or system security officers, and other security relevant events. For each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event. For identification/authentication events the origin of request (e.g., terminal ID) shall be included in the audit record. For events that introduce an object into a user's address space and for object deletion events the audit record shall include the name of the object. The ADP system administrator shall be able to selectively audit the actions of any one or more users based on individual identity. [Ref. 10, p. 16]

The actions of a user should be traceable by the use of an audit trail. It should be possible to bind a subject or user to all actions noted above or that are deemed security relevant by the proper authorities so as to make the users of the system accountable for their actions. In Class C2 access control protection the following events should be recorded at a minimum [Ref. 13, p. 9]:

- Use of Identification and authentication mechanisms
- Introduction of objects into a user's address space
- Deletions of Objects from a user's address space
- Actions taken by computer operators and system administrators and/or system security administrators
- Production of printed output
- All other security events as defined by the system administrators

In each of the preceding events it must be possible to associate an authenticated user ID with each event. Software used to perform the audit as well as the audit trail itself, should be protected by the TCB and should be subject to strict access controls. The security requirements of the audit mechanism are the following [Ref. 14]:

- The event recording mechanism should be part of the TCB and as such protected from unauthorized modification or circumvention.
- The audit trail must be protected by the TCB and accessible only to audit personnel. It too must be protected from unauthorized modification.
- The ability to enable or disable the event recording mechanism should be a part of the TCB and remain inaccessible to unauthorized users.

One way to protect the audit trail in addition to protecting the audit files by DAC is to record them on a device that is designed to be a *write-only* device or by setting the designated device to *write-only-once* by disabling the read mechanism. Modification of data already recorded would be quite difficult as an attacker would not be able to go back and read or find the data that they wish to modify. However, a drawback to this method is that the medium must be switched over to a read device introducing a time delay before the audit trail can be examined. One way to offset this is to pass copies of all audit records to the system security administrator as they are sent to the write-only device. However, this medium must be protected at the highest levels afforded by the system. [Ref. 13, p. 15]

D. OPERATIONAL ASSURANCE

1. System Architecture

> The TCB shall maintain a domain for its own execution that protects it from external interference or tampering (e.g., by modification of its code or data structures). Resources controlled by the TCB may be a defined subset of the subjects and objects in the ADP system. The TCB shall isolate the resources to be protected so that they are subject to the access control and auditing requirements. [Ref. 10, p. 16]

Although systems at the Class C2 level need not be specifically designed for security, they must support sound principles of hardware and operating system design, as well as support specific security features. Resources should be protected so that they are subject to access control and auditing. Examples include putting an access control list on the password file and the protection of user files so that they are not accessed by other users through accident or design. Privileged programs (such as the auditing program) should not be interfered with by user programs [Ref. 15, pp. 134-135].

2. System Integrity

> Hardware and/or software features shall be provided that can be used to periodically validate the correct operation of the on-site hardware and firmware elements of the TCB. [Ref. 10, p. 17]

"System integrity means that the hardware and firmware must work and be tested to ensure that they keep working." [Ref. 15, p. 136] Some vendors meet the requirement for a system integrity test by providing a set of integrity tests which are conducted as a regular system exercise whenever the system is powered up. If the tests fail the system will not *boot.* Diagnostics are normally performed during preventive maintenance periods. [Ref. 15, p. 137]

E. SECURITY TESTING

> The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. Testing shall be done to assure that there are no obvious ways for an unauthorized user to bypass or otherwise defeat the security protection mechanisms of the TCB [Ref. 16]. Testing shall also include a search for obvious flaws that would allow violation of resource isolation, or that would permit unauthorized access to the audit or authentication data. (See the Security Testing guidelines.) [Ref. 10, p. 17]

The system should be tested to see if it meets its stated security capabilities. Particular attention should be paid to the "Identification and Authentication" mechanism as this is normally the first line of defense in any TCB. Additionally it should not be possible for an unauthorized user to access or alter audit information and authentication data. Systems meeting the requirements of Class C2 should provide some protection against human error and preventing and detecting user abuse of authority and direct probing. [Ref. 17, p. 6]

F. DOCUMENTATION

1. Security Feature User's Guide

A single summary, chapter, or manual in user documentation shall describe the protection mechanisms provided by the TCB, guidelines on their use, and how they interact with one another. [Ref. 10, p. 17]

2. Trusted Facility Manual

A manual addressed to the ADP system administrator shall present cautions about functions and privileges that should be controlled when running a secure facility. The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given. [Ref. 10, p. 17]

3. Test Documentation

The system developer shall provide to the evaluators a document that describes the test plan, test procedures that show how the security mechanisms were tested, and results of the security mechanisms' functional testing. [Ref. 10, 17]

4. Design Documentation

Documentation shall be available that provides a description of the manufacturer's philosophy of protection and an explanation of how this philosophy is translated into the TCB. If the TCB is composed of distinct modules, the interfaces between these modules shall be described. [Ref. 10, p. 17]

IV. CONTROLLED ACCESS PROTECTION IN TELESCRIPT™

A. SETTING THE STAGE

Before examining controlled access protection in Telescript™ it is essential to understand that although the language and its execution mechanism, the engine, incorporate security features that will be discussed in this chapter they were not designed to act as trusted software. Accordingly, the Telescript™ system does not explicitly enforce any security policy and has no stated security policy model. However it is possible to devise a simple model that the Telescript™ could enforce using discretionary features (classes) built into the language.

In the analysis that follows the Telescript™ language and execution mechanism are examined to determine to what degree they meet the minimal requirements for systems assigned a Class (C2) rating as defined in the Department of Defense Trusted Computer System Evaluation Criteria [Ref. 10] and explained in Chapter III.

B. BUILDING A USER-DEFINED SECURITY POLICY

1. Discretionary Access Control

As described previously the Telescript™ environment is essentially a process (the engine) that runs on top of the host machine. Associated with an engine is an engine place where all other processes (threads) are executed by the engine. Essentially the engine only executes two types of processes: agents and places.

The system administrator builds this environment through a script which is actually a collection of classes or objects that will be active in the engine place under the system administrator's authority. Generally these objects perform services for host subscribers through agent-to-agent or agent-to-place interaction. These services may or may not require access to the host applications (e.g. databases) or file systems depending on the system's configuration and the service provider's desires. It all depends on the nature of the service being provided. It is entirely possible that services can be provided that require no access to the host system. When the process (engine) is booted all the places and non-mobile agents (which typically provide services) that will exist within the engine place are created as threads that are more or less permanent in duration. Once the engine process is functional the administrator performs operations, administration and management (OA&M) functions through the Thumper™ panel.

All interaction in the environment is between objects (in the object-oriented sense of the word) and is sponsored by the engine in that the engine evokes the 'live' method on each object (process or more correctly thread). Thus, some degree of protection is provided by object-oriented nature of the language. Object features are made *public* or *private* at the discretion of the service provider. This affects the get (read) and set (write) functions common to all objects regarding their attributes. Additionally, the engine can provide protected references between objects making the referenced objects read-only.

Objects can also inherit features from classes that are specifically designed for security. These will be described in more detail later.

To examine Telescipt in terms of a security policy of controlled access protection it is necessary to design a rudimentary security model based on the following assumptions:

- First, all processes (threads) that are created in the engine place by the service provider at boot-up (places and non-mobile agents) shall be considered named objects. However, these objects will act as subjects when performing actions at the request of either subjects under system authority or subjects (agents) not under system authority.
- Second, any processes (threads) or objects created in the engine space at behest of the region authority (again, the service provider) will be considered named objects. These objects may also act as subjects as outlined in first assumption.
- Third, any host system applications or files will be considered named objects. These objects may not act as subjects.
- Fourth, mobile agents that enter the engine space after proper Identification and Authentication or are created by subjects that have entered and occupy any places within the engine place are named subjects.

- Fifth, subjects that are not of region authority (agents) may not interact with each other. The reason for this restriction will become more apparent later in this discussion.
- Sixth, subjects access named objects (places) by 'entering' them (being passed a reference to the place by the engine) or by having pre-existing or created objects passed to them by places (acting as subjects) that they are interacting with contingent to the restrictions outlined in the fifth assumption. This restriction is necessary because a place may be considered a named subject when passing messages to or performing functions on behalf of an agent.

Therefore, access controls are placed on agents (subjects) not of the region's authority that wish to access objects in the service provider's engine place. To enforce access controls all objects and subjects must be named and this is accomplished by a unique identifier telename which denotes the authority or owner of the process. The notion of group controls is not supported. Public controls are supported constructing a place that inherits from the *AC Public Class* (A subgroup of the *Access Control Group* described below). Such places can be made accessible by all agents.

Access control between the agents and places is done creating places that inherit from the *Access Control Group* of classes. All classes in this group restrict access to places created from them to one or more authorities (*telenames*) and have a requirement for a degree of trust in the authentication system. The degree of trust is interpreted relative to the policy set by the local region and is system dependent. Additionally the place may require that an agent (subject) provide *credentials* to access the object (place or non-mobile agents). These credentials are provided to individual users by the service provider and are data elements known that should be known only by the two of them. In effect an access control list can be attached to each object operation that specifies the authorities that can use each operation. Propagation of access rights can be prevented by disallowing subjects not under system authority to interact with each other. This is necessary because the current Telescript™ language does not prevent a subject with the proper authority and credentials from passing object references to subjects that do not have full access rights to the objects in question nor does it prevent subjects from evoking operations of other subjects and using the called subject's operations (the called object becomes the sponsor of this operation which is called a sponsored operation) and access rights. The calling subject could *masquerade* as the called subject. However, as subjects have the ability to protect their attributes and operations, such an attack would have to be cooperative. Remember that interaction between subjects not under system authority would require them to occupy the same place. As the language is currently designed, both of the subjects could gain entry to the same place but with different access rights to the operations of the place. If subjects do not possess the appropriate access rights an exception is thrown and entry to a place is denied.

Agents also carry *native* permits that are assigned by the agent's author (*authority*). The engine place has a *regional permit* and all other *places* have a *local* permit. The regional and local permits are granted by the service provider and limit the actions of agents that enter both the engine place and all other places as appropriate. These permits limit the following:

- age: maximum thread age in seconds.
- extent: maximum thread size in octets.
- priority: maximum priority of thread for scheduling by the engine.
- canCreate: True if the thread can create new processes.
- canGo: True if the thread can evoke the *go* operation to travel.
- canGrant: True if a thread can increase the permit of threads it has created.
- canDeny: True if a thread can decrease the permit of threads it has created.

Permits are primarily concerned with resource consumption on the host platform and controlling the capabilities of threads. They are not focused on access control but they can support a security policy that tries to prevent denial of service. It is also important to note that no agent can create another agent with greater access to objects than it possesses itself. The effective permit of a thread is the intersection of the native, local and regional permits and must be re-negotiated every time an agent enters a new place.

An agent can attempt to gain access to a place by presenting a petition to the engine. This petition notifies the engine of the place that it wants to visit by *teledress* and what services are required there. The engine then evokes the entering operation on the desired place. If the agent meets the access requirements and possesses the proper permit the engine evokes the agents *live* method and passes a reference to the place to the agent and a reference from the agent to the place. The two threads are then free to call each others operations and attributes. Note: Access to host system objects can only be done by evoking the operations of objects. Subjects are never given direct access to the host system. If the entering operation fails the agent is either terminated, requests to enter a different place, or may be sent to another engine via the *go* operation. Its final fate depends on the quality of exception code written into the agent.

In sum, all interprocess (thread) access is mediated by the Telescript™ engine. Operating as a virtual machine atop the host operating system the engine acts similarly to a system kernel isolating the threads that it controls from the host.

> The enforcement mechanism (e.g., self/group/public controls, access control lists) shall allow users to specify and control sharing of objects by named individuals, or defined groups of individuals, or by both, and shall provide controls to limit the propagation of access rights. [Ref. 10, p. 15]

The primary enforcement mechanism of Telescript™ is the standard protection afforded by the object-oriented nature of the language and how it is used with the engine. Objects are not created at compilation, as the language is interpreted and all initial objects are instantiated when scripts are submitted to the engine at boot up. The telename attribute of each subject and object uniquely names each. Within objects, a subject's access to operations and attributes is controlled by the use of individual access control lists on each. The access lists contain the telenames and credentials of subjects granted access to the pertinent operation or attribute. This restriction is quite substantial as it restricts access to authorities that are known in advance. The access control lists themselves are attributes of the subjects. Subject attributes are protected by making them private data structures, in the object-oriented sense, within the subject.

Objects in the engine space that are neither agents nor places, i.e. data structures or created objects, can be protected by making references to them protected (read-only). Additionally, although all objects created by function call, or read, could have an access list attached to them, this would be unnecessary as they would not be passed to subjects that did not have the proper authority or credentials to make the function calls or reads.

The propagation of access rights to between named subjects not under system authority cannot be prevented unless the subjects are prohibited from interacting with each other as stated the previous assumptions. Subjects that call on objects gain the access rights of the objects only if they are a named subject and possess the proper credentials.

> The discretionary access control mechanism shall, either by explicit user action or by default, provide that objects are protected from unauthorized access. These access controls shall be capable of including or excluding access to the granularity of a single user. [Ref. 10, p. 15]

Objects are explicitly protected from unauthorized access to the granularity of a single user by including (granting) access as identified by a telename with the proper credentials. There is no explicit attribute in the *Access Control Group* of classes to exclude access to a single user.

"Access permission to an object by users not already possessing access permission shall only be assigned to authorized users." [Ref. 10, p. 15]

This can be done in the Telescript™ environment, but only at great risk and only by another subject not under system authority. Surprisingly, this would violate system integrity. As a matter of explanation, if a subject (agent) in Telescript™ does not have access rights to an object (place) or, its functions and attributes, it should not be assigned them by another subject as this would be an unauthorized propagation of access rights. Speculatively, if a change in access permission was desired, the engine or subjects under its region authority would have to alter the access control list attributes internal to the object that the agent wishes to access. Although this can be done by submitting a new script for the object, this is not a normal operating procedure during process (engine) execution. Regardless, the regional authority can not change the credentials of the requesting agent as this attribute would normally be a private attribute of the agent created when the agent is instantiated at its host system. Agents may propagate their own credentials to any clones they create if the effective permit in the engine space that they occupy allows them to clone themselves. Additionally, the credentials of clones created by agents would be copies of those possessed by the agents. Propagation of access rights is not a normal event in an executing Telescript™ process that is trying to enforce access control.

In summary the propagation of access rights between subjects in Telescript™ can be enforced by the security properties of the language and engine if the interaction between the subjects not under system authority is proscribed as outlined in previous assumptions on building a user-defined security policy.

2. Object Reuse in Telescript™

> No information, including encrypted representations of information, produced by a prior subject's actions is to be available to any subject that obtains access to an object that has been released back to the system. [Ref. 10, p. 15]

In Telescript™, a module, conceivably similar to that in Lisp, automatically keeps track of inaccessible blocks and makes them available to be reallocated by returning dynamic storage to the heap or permanent storage to the file access table [Ref. 18, p. 683]. Objects that are created by a subject in any storage medium are reclaimed by garbage collection after the subject is terminated. Any objects owned by the subject are de-referenced and the medium is released for use by other subjects. The objects are not completely destroyed and the possibility of reconstruction is not eliminated as the physical space occupied by the objects is not overwritten. Hence the object reuse requirement is not met by Telescipt.™

C. ACCOUNTABILITY

1. Identification and Authentication (I&A)

Agents go from place to place by traveling. To travel an agent provides a *ticket* to the engine of its current region. The ticket contains the telename of an engine in a region that the agent wishes to visit. If the agent possesses the proper permit as discussed above it is encoded by the sending engine into a what is called a *bag of bits* and transported to the its destination via the available communications infrastructure (CI). Engine to Engine transfer of the agent uses an *authentication regime* that requires strong mutual authentication using RSA public key encryption [PKCS], session key negotiation using the Diffie-Hellman algorithm with perfect forward security [DIFFIE], and session encryption using RC4. [Ref. 19] When the agent is decoded it is checked for a valid telename which identifies its authority or owner. If the *telename* is valid and the agent has the proper regional permit the agent is given a thread of execution. Normally, the agent will request an entering or meeting operation as discussed previously. The agent will present credentials which act as a token for all intents and purposes. The places protect the authentication data making the data a private attribute and the operations used to access the data private in a object-oriented manner similar to C++ or Ada.

Each authority's agents can be uniquely identified by a telename and by an internal attribute similar to a process ID when more than one agent of an authority is present in a region. Each agent's actions can be audited. This will be described in the audit section in greater detail.

> The TCB shall require users to identify themselves to it before beginning to perform any other actions that the TCB is expected to mediate. Furthermore, the TCB shall use a protected mechanism (e.g., passwords) to authenticate the user's identity. The TCB shall protect authentication data so that it cannot be accessed by any unauthorized user. [Ref. 10, p. 16]

In Telescript™ the agent's sending engine is authenticated prior to identification and authentication of the agent. If the sending engine is authenticated then the agent is identified and authenticated by its telename (an octet string, issued by General Magic, or the service provider) which names the process (thread) and denotes on whose behalf it operates (authority). The authentication data tables are accessible to the engine process only.

> The TCB shall be able to enforce individual accountability by providing the capability to uniquely identify each individual ADP system user. The TCB shall also provide the capability of associating this identity with all auditable actions taken by that individual. [Ref. 10, p. 16]

Each agent (subject) is uniquely identified by telename which is associated with all events generated by the logging mechanism in the *event record.*

The Identification and Authentication mechanism in Telescript™ meets the requirements for Class C2 systems.

2. Audit

If a subject's request to enter a place is successful the operation returns an object that is a record of occupancy for the place. Using the Thumper™ panel the system administrator can determine the occupants of any place. Places keep a list of occupants that is updated upon entry and departure of agents. The creation and maintenance of an audit trail of all subject accesses can achieved if places inherit from the *Logged* and *Event* classes. Logging is implemented using *event collection points*. These objects transfer entries to external databases through an external log server that is not accessible by subjects in the system. All reportable events are recorded using the *Current Process Event Id* (thread ID) which includes the process telename responsible for the event or action. [Ref. 20] Reportable events are declared in an *Event Record* or *OAM Single Event* object that is created at engine boot-up. If places inherit from the *AC Logged* class, changes in that place's access controls and attempts to enter the place without the required access rights opens a log entry.

For run time evaluation of the logging system the Thumper™ *panel* is invaluable. Using the panel the system administrator can determine [Ref. 9]:

- The specific event records and which specified logs they are being dumped to.
- Open a new event record and log.
- Stop logging a specific event record.
- List event records that are generated but are not being logged.
- Perform additional administrative tasks as required (e.g. adding and removing subscribers from the service, restarting engines, killing clouds, etc.).

The audit logs are examined off-line and an audit trail can be established for each thread based on telename included in the *Current Process Event* Id. The standard log entry is a six-tuple that looks like this: *<applicationName, transactionId, authority, identity, classDigest, current-time>* where applicationName is the subject by *telename*, transactionId is the event, authority is as explained elsewhere, classDigest is related to the classes carried in the object (object-oriented sense), and current-time is exactly as stated.

> The TCB shall be able to create, maintain, and protect from modification or unauthorized access or destruction an audit trail of accesses to the objects it protects. The audit data shall be protected by the TCB so that read access to it is limited to those who are authorized for audit data. [Ref. 10, p. 16]

The engine controls access to the event (audit) record for all objects (object-oriented sense) that have inherited from the *Logged* and *Event* classes through an external log server. The log server should be accessible only by those authorized for audit data but this is not within the province of Telescript™. Administration of the logs can be controlled from the Thumper™ panel as detailed above but detailed analysis is conducted off-line.

> The TCB shall be able to record the following types of events: use of identification and authentication mechanisms, introduction of objects into a user's address space (e.g., file open, program initiation), deletion of objects, actions taken by computer operators and system administrators and/or system security officers, and other security relevant events. For each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event. [Ref. 10, p. 16]

Telescript™ can log all events prescribed in the proceeding paragraph if the Event Record and OA&M Single Event objects are configured properly. The existence of a six-tuple in the event record as described above is evidence of the success of the event. Event failures would be logged in an event record that inherits *from OAM Access Control Violation* (for attempted unauthorized access) or *OAM Alarm Event* for other failures.

> For identification/authentication events the origin of request (e.g., terminal ID) shall be included in the audit records. For events that introduce an object into a user's address space and for object deletion events the audit record shall include the name of the object. The ADP system administrator shall be able to selectively audit the actions of any one or more users based on individual identity. [Ref. 10, p. 16]

For identification/authentication requests the origin of the request will always be within the engine space. Created or deleted objects can be reflected in the event log. The ADP system administrator can selectively audit the actions of any one or more users based on the individual identity provided by the *telename* field of the six-tuples in the event record.

If the Event Record and OA&M Single Event objects are properly configured by the system administrator Telescript™ can meet the Class C2 Audit requirements.

D. OPERATIONAL ASSURANCE

1. System Architecture

"The TCB shall isolate the resources to be protected so that they are subject to the access control and auditing requirements." [Ref. 10, p. 16]

The engine executes in a *domain* (the engine space) that appears to be secure from interference or tampering if the engine is executed as a privileged process. Yet, the engine domain can not be said to be secure with any degree of assurance.

The deliberate decision to interpret rather than execute agent code serves to protect the engine from errant or malicious code to some degree as undefined classes are not instantiated. With this limitation, code executed must be within the bounds of a defined class under the restrictions of the class definition. Code not recognized by the engine is not executed.

The protection of resources controlled by the engine is more problematic. If the subjects, as defined above, are prevented from interacting with each other, there is little likelihood that misbehaved agents will gain access to resources that they are not entitled to. All resources in the domain can be isolated and are subject to access control and auditing. Therefore, the engine can create domains internal to the engine space.

It cannot be stated with any degree of assurance that Telescript™ meets the Class C2 Operational Assurance criteria for System Architecture. This is unlikely to change without examination of the design documentation.

2. System Integrity

> Hardware and/or software features shall be provided that can be used to periodically validate the correct operation of the on-site hardware and firmware elements of the TCB. [Ref. 10, p. 17]

The Telescript™ Developer's Kit comes with scripts that can be used to boot engines and create places of limited complexity, capability and utility. The operations of these can be checked using the Thumper™ panel and supplied on-line debugger. Scripts could be submitted to the engine which create agents that test software features of the Telescript™ environment that are security relevant.

More needs to be done before Telescript™ can satisfy this requirement.

E. SECURITY TESTING

"The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation." [Ref. 10, p. 17]

Although Telescript™ appears to be designed with security considerations in mind, particularly through the inclusion of many classes that relate to access and logging, General Magic makes no claims that the current engine is completely safe. To date, the engine implementation has not been through a rigorous security verification or certification program. Furthermore, analysis in this thesis has been restricted to a rigorous examination of the language and supporting documentation and is not based on quantitative tests running the engine on hardware. Scripts could be written to test security mechanisms of the ADP system as stated above.

F. DOCUMENTATION

1. Security Feature User's Guide

> A single summary, chapter, or manual in user documentation shall describe the protection mechanisms provided by the TCB, guidelines on their use, and how they interact with one another. [Ref. 10, p. 17]

The Telescript™ Programming Guide includes a section on writing secure applications [Ref. 21, pp. 75-79] and the Telescript™ Application Library [Ref. 20] is an excellent guide to the security classes defined in the language. Additionally there is a General Magic White Paper, "An Introduction to Safety and Security in Telescript" [Ref. 19] which provides further guidance on safety and security considerations in Telescript™ programming.

2. Trusted Facility Manual

> A manual addressed to the ADP system administrator shall present cautions about functions and privileges that should be controlled when running a secure facility. [Ref. 10, p. 17]

There is no Trusted Facility Manual for Telescript™ programming.

3. Test Documentation

See Security Testing section above.

4. Design Documentation

> Documentation shall be available that provides a description of the manufacturer's philosophy of protection and an explanation of how this philosophy is translated into the TCB. [Ref. 10, p. 17]

The manufacturer's philosophy of protection could only be inferred from the limited security material that was available. Since protection was described as relating to various levels of 'paranoia' it can only be assumed that the primary emphasis of the manufacturer was on making the language communications-oriented and focuses on the novel attraction of moving the client (agent) to remote servers (engines). In fairness however, the use of interpreted code, and the security features of object-oriented programming were stressed and some degree of protection is provided through security classes whose use is purely discretionary.

V. CONCLUSION

A. SECURITY POLICY ENFORCEMENT IN TELESCRIPT™

This thesis has provided an initial investigation into the capability of the Telescript™ language and execution mechanism, the engine, to meet the Class C2 requirements for discretionary access control. To accomplish this an examination of the documentation provided by General Magic in their Telescript™ Development Kit (TDK) and Active Web Tools™ (AWT) was conducted. The results of this review were correlated with the requirements put forth in the Department of Defense Trusted Computer System Evaluation Criteria [Ref. 10]. A more thorough investigation into the properties of the Telescript™ language and engine would be difficult to accomplish without access to the proprietary source code.

The General Magic concept of mobile agents and Remote Programming (RP) as proposed by Jim White [Ref. 1] is an exciting one and is already being used in Magic Mail™ personal communication applications but serious security issues remain unsolved. The commercial novelty and benefits of using mobile agents has over-shadowed security concerns.

This is not to say that security has been completely overlooked in Telescript.™ However, the current version of Telescript™ does not meet all of the Class C2 requirements for discretionary access control. By using the Telescript™ language's built-in Access Control Class it is possible to limit the propagation of access rights between subjects if a user-defined security policy is built using the assumptions outlined in Chapter IV. The requirement for accountability in terms of identification and authentication and audit can be met using the Telescript™ language's built-in classes. The Telescript™ language and execution mechanism can not be said to be operationally secure in terms of system architecture and integrity with any degree of assurance. Finally, security testing needs to be done and security documentation is lacking.

B. FUTURE RESEARCH

Follow-on work to this thesis should include testing the Telescript™ execution mechanism (engine) by:

- Setting up a stand alone network as a region under a single authority with engines running each individual machine in the network.
- Creating places in each engine space, each with different access requirements. Some places would access system resources.
- Building agents of different authority that attempt to penetrate places and obtain references to objects or system resources that they do not have the access rights to.

Similar work should be done on other languages that support mobile agent like behavior. Candidates for this work include Java™ and TCL.

LIST OF REFERENCES

1. White, Jim. *Telescript Technology: Mobile Agents*, General Magic White Paper, Sunnyvale, California: General Magic, 1995.

2. White, Jim. *Telescipt Technology: An Introduction to the Language*, General Magic White Paper, Sunnyvale, California: General Magic, 1995.

3. Huer, Karl. *Agents*, http://www.pcug.org.au/~kauer/project/ (23 Dec. 1995).

4. Jennings, Nicholas and Woolbridge, Michael. "Agent Theories, Architectures, and Languages: a Survey." Woolbridge and Jennings Eds., Intelligent Agents, Springer-Verlag, Vol. 1-22, 1995.

5. White, J. E. "A high-level framework for network-based resource sharing. " *Proc. AFIPS Conference 45*, 1976.

6. General Magic, Inc. *Telescript Language Reference*, Sunnyvale, California, October 1995.

7. General Magic, Inc. *Telescript Language Lessons*, Sunnyvale, California, 1995.

8. General Magic, Inc. *Telescript Language Reference Manual*, Sunnyvale, California, 1995.

9. General Magic, Inc. *Thumper User's Guide*, Sunnyvale, California, 1995.

10. National Computer Security Center. *DoD Trusted Computer System Evaluation Criteria*, Department of Defense, DoD 5200.28-STD, December 1985.

11. National Computer Security Center. *A Guide To Understanding Discretionary Access Control in Trusted Systems*, NCSC-TG-003, Ver. 1, 30 September 1987.

12. National Computer Security Center. *A Guide To Understanding Identification And Authentication In Trusted Systems*, NCSC-TG-017, September 1991.

13. National Computer Security Center, *A Guide to Understanding Audit In Trusted Systems*, NCSC-TG-001 Ver. 2, 1 June 1988.

14. Gligor, Virgil D. *Guidelines for Trusted Facility Management and Audit*. University of Maryland, 1985.

15. Gangemi, G. T. Sr. and Russell, Deborah. *Computer Security Basics*. Sebastopal, California: O'Reilly & Associates, Inc., 1991.

16. Department of Defense. *ADP Security Manual - Techniques for Implementing, Deactivating, Testing, and Evaluating Secure Resource-Sharing ADP Systems*, Department of Defense, DoD 5200.28-M, June 1979.

17. Brinkley, Donald L., and Schell, Roger R. *Evaluation Criteria For Trusted Systems*, in Information Security; An Integrated Collection of Essays, eds. Abrams, Gajodia, & Podell, IEEE Computer Society Press, Los Alamitos, California, 1995.

18. Feldman, M., Koffman, E. *Ada Problem Solving and Program Design*, New York: Addison-Wesley Publishing Company, 1993.

19. General Magic Inc. *An Introduction to Safety and Security in Telescript*, Sunnyvale, California, 1995.

20. General Magic Inc. *Telescript Application Library*, Sunnyvale, California, 1995.

21. General Magic Inc. *Telescript Programming Guide*, Sunnyvale, California, 1995.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
Naval Postgraduate School
411 Dyer Rd.
Monterey, CA 93943-5101

3. Chairman, Code CS 2
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

4. Dr. Cynthia E. Irvine, Code CS/IC 6
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

5. Louis D. Stevens, Code CS/ST 2
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

6. General Magic 1
ATTN: Hiu Ng
420 North Mary Avenue
Sunnyvale, CA 94086

7. Dr. Blaine Burnham... 1
R23
National Security Agency
9800 Savage Road
Fort Meade, MD 20755-6000

8. SPAWAR PD 71M..1
ATTN: Capt. Mary A. Shupack
Program Manager Information Security
2451 Crystal Drive
Arlington, VA 22245-5200

9. Lcdr Robert L. Marlett 4
USCINCPAC/J21
Box 64010
Camp Smith, HI 96861-4010